Spring Framework

Spring Overview

- Spring is a Lightweight ***Application*** Framework
- Spring Promotes loose coupling through ***Inversion of Control (IoC)***
- Spring comes with rich support for

  Database support

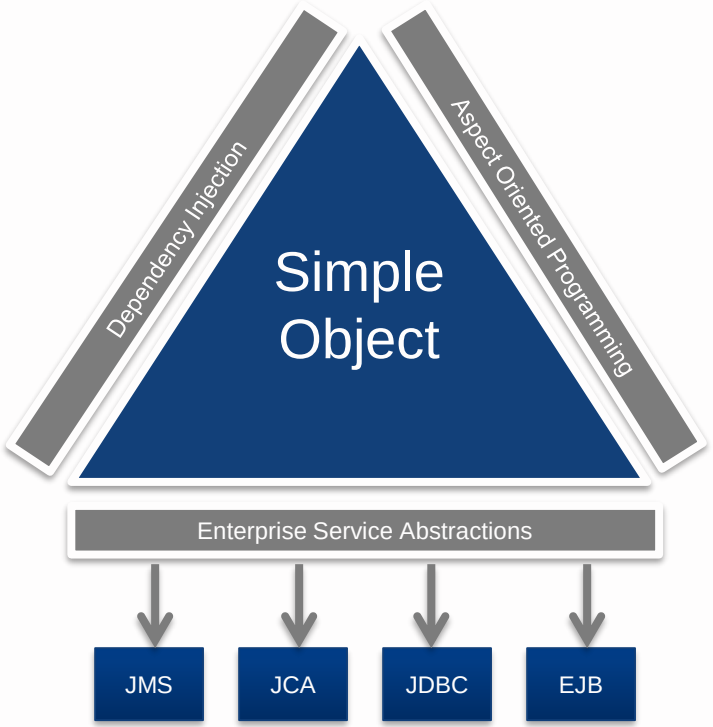
  Hibernate/JPA

  Transaction management

  MVC

  REST WebServices

The Spring Triangle



Why Use Spring?

- Wiring of components (Dependency Injection)
  - Promotes/simplifies decoupling, design to interfaces
  - Declarative programming
  - Easily configured aspects, esp. transaction support

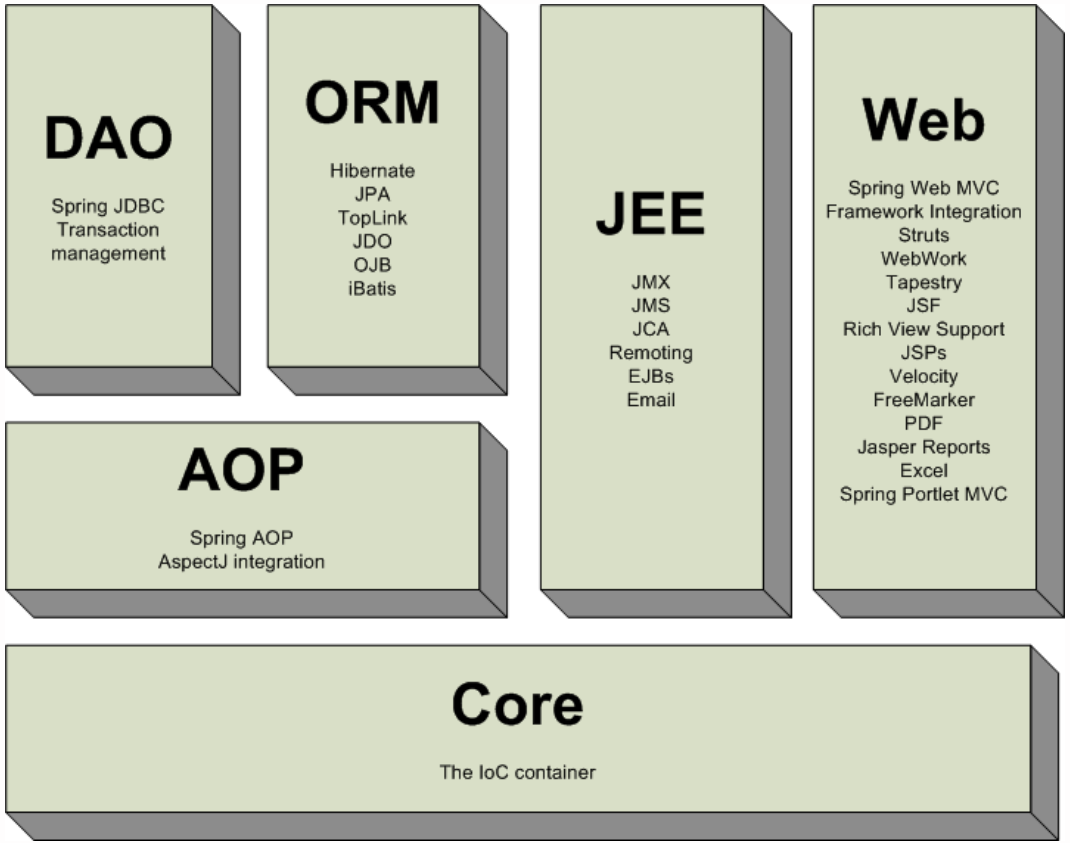
Why Use Spring?

- Conversion of checked exceptions to unchecked
- Not an all-or-nothing solution
  - Extremely modular and flexible
- Well designed
  - Easy to extend
  - Many reusable classes

Spring Benefits

- Spring can effectively organize your middle tier objects, whether or not you choose to use EJB
- Spring's configuration management services can be used in any architectural layer, in whatever runtime environment
- Spring can use AOP to deliver declarative transaction management without using an EJB container
- Spring provides a consistent framework for data access, whether using JDBC or an O/R mapping product such as TopLink, Hibernate
- Spring provides a consistent, simple programming model in many areas JDBC, JMS, JavaMail, JNDI and many other important API’s

Spring modules



Spring and Dependency Injection

- Inversion of Control (IoC)
- “Hollywood Principle”
  - Don't call me, I'll call you
- “Container” resolves (injects) dependencies of components by setting implementation object (push)
- As opposed to component instantiating or Service Locator pattern where component locates implementation (pull)
- Martin Fowler calls it as Dependency Injection

IoC Container

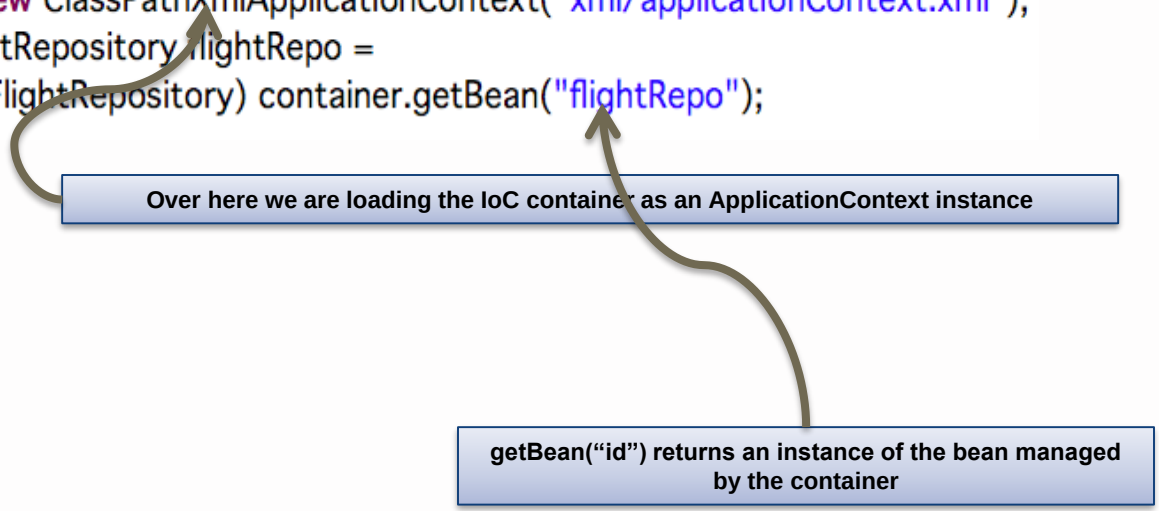
- The term *Container* which is commonly used in EE, is also used by Spring and is referred to as an IoC (Inversion of Control) Container
- Spring provides a Container/Factory/Context which manages
  - Component instantiation and initialization
  - Component dependencies
  - Services wrapped around those Components
- Different types of DI techniques
  - Setter injection
  - Constructor injection
  - Field injection

Loading the IoC Container

- **BeanFactory** is the actual container managing all bean instances
- **ApplicationContext** derives from BeanFactory and extra support for:
  - Support for I18N, via MessageSource
  - Access to resources, such as files and URLs
  - Event propagation
  - Loading of multiple contexts
- **WebApplicationContext** is further derived from ApplicationContext. To be used in a web application

Example

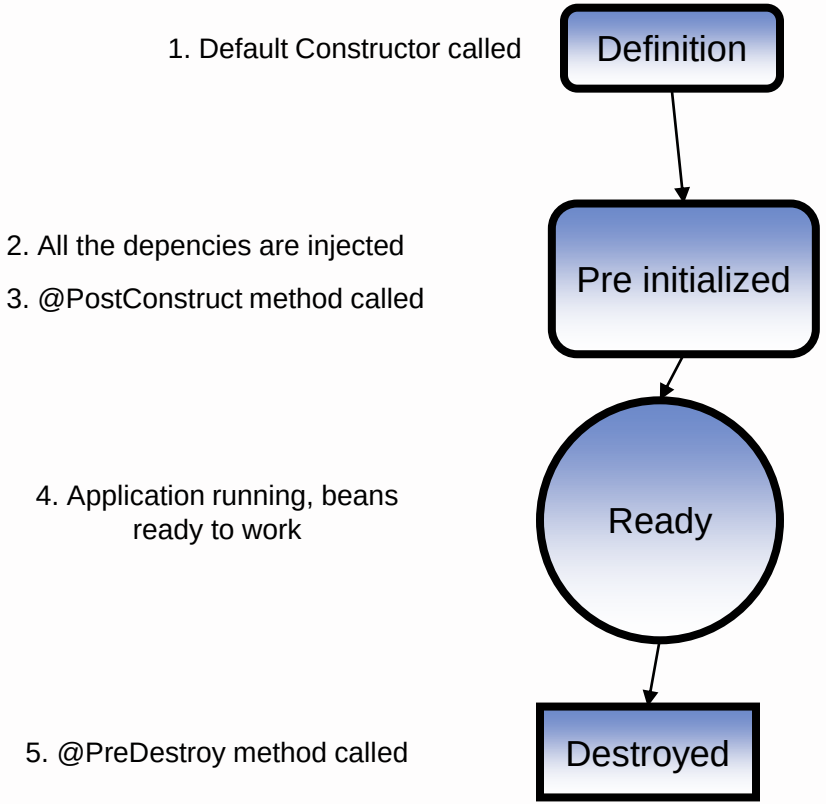
```
ApplicationContext container =
    new ClassPathXmlApplicationContext("xml/applicationContext.xml");
FlightRepository flightRepo =
    (FlightRepository) container.getBean("flightRepo");
```



Scope of a bean

- Beans managed by the container can have different scopes:
  - singleton (default)
  - prototype (non singleton)
  - request (Spring MVC)
  - session (Spring MVC)

Bean lifecycle



Annotations based life cycle callback

```
public class LifecycleBean {

    @PostConstruct
    public void setup() {
            //some custom initialization
    }

    @PreDestroy
    public void cleanup() {
            //some custom cleanup
    }
}
```

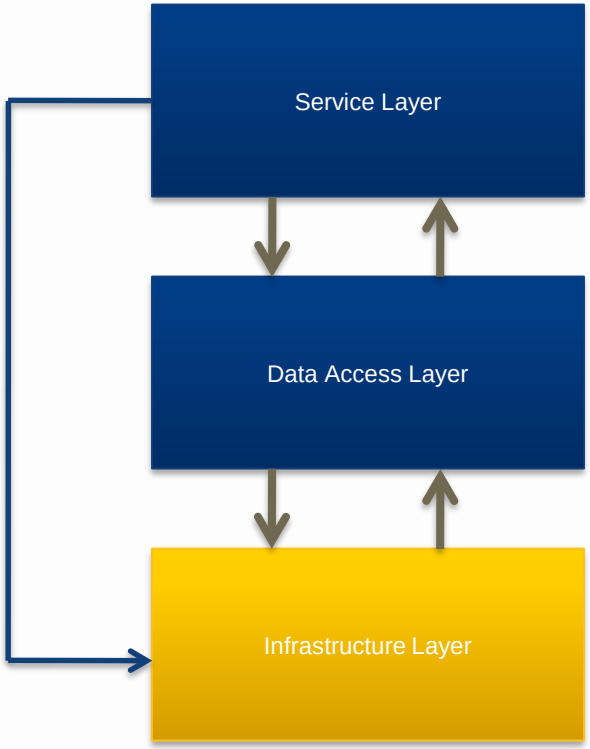
LTI
Let's Solve

Celebrating 20 Years

Business layer of an application

Larsen & Toubro
Group Company

Layered Architecture



The Service Layer

- Defines the public functions of the application
  - Clients call into the application through this layer
- Encapsulates the logic to carry out each application function
  - Delegates to the infrastructure layer to manage transactions
  - Delegates to the data access layer to map persisent data into a form needed to execute the business logic

The Data Access Layer

- Used by the service layer to access data needed by the business logic
- Encapsulates the complexity of data access
  - The use of data access API
    - JDBC, Hibernate, etc
  - The mapping of data into a form suitable for business logic
    - A JDBC ResultSet to a domain object graph

The Infrastructure Layer

- Exposes low-level services needed by other layers
  - Infrastructure services are provided by Spring
  - Developers typically do not write them
- Likely to vary between environments
  - Production vs. test

Layers working together

- Client calls a function of the service layer
- A service initiates a function of the application
  - By delegating to the transaction manager to begin a transaction
  - By delegating to repositories to load data for processing
    - All data access calls participate in a transaction
    - Repositories oftern return domain objects that encapsulate domain behaviors

Cont’d…

- A service continues processing
  - By executing business logic
  - Often by coordinating between domain objects loaded by repositories
- And finally, completes processing
  - By updating changed data and committing the transaction

LTI
Let's Solve

Celebrating 20 Years

Spring support for Database access

Larsen & Toubro
Group Company

Role of Spring while accessing database

- Enable a layered application architecture
  - To isolate an application's business logic from the complexity of data access
- Spring manages resources for you
  - Eliminates boilerplate code
  - Reduces likelihood of bugs
- Declarative transaction management
  - Transaction boundaries declared via configuration
- Automatic connection management
  - Connections acquired/released automatically
- Intelligent exception handling

Spring’s JdbcTemplate

- Greatly simplifies use of the JDBC API
  - Eliminates repetitive boilerplate code
  - Alleviates common causes of bugs
  - Handles SQLExceptions properly
- Without sacrificing power
  - Provides full access to the standard JDBC constructs

Introduction to JdbcTemplate API

```
public int getTotalFlights() {
        return jdbcTemplate.queryForObject("select count(*) from flights_test", Integer.class);
}
```

- Acquisition of the connection
- Participation in the transaction
- Execution of the statement
- Processing of the result set
- Handling any exceptions
- Release of the connection

Spring JDBC – Who does what?

Action	Spring	You
Define the connection parameters		X
Open the connection	X	
Specify the SQL statement		X
Declare parameters and provide parameter values		X
Prepare and execute the statement	X	
Setup the loop to iterate through the results (if any)	X	
Do the work for each iteration		X
Process any exception	X	
Handle transactions	X	
Close the connection, statement and resultset	X	

Example on performing a select operation

```
public List<Flight> getAvailableFlights(String carrier) {
 class FlightMapper implements RowMapper<Flight> {
     @Override
     public Flight mapRow(ResultSet rs, int index) throws SQLException {
          Flight flight = new Flight();
          flight.setFlightNo(rs.getString(1));
          flight.setCarrier(rs.getString(2));
          flight.setFrom(rs.getString(3));
          flight.setTo(rs.getString(4));
          return flight;
     }
 }
 return jdbcTemplate.query(
 "select * from flights where carrier = ?", new FlightMapper(), carrier);
}
```

Example on performing a DML operation

```
public void newFlight(Flight flight) {
        jdbcTemplate.update(
                "insert into flights_test values(?, ?, ?, ?)",
                        flight.getFlightNo(), flight.getCarrier(),
                        flight.getFrom(), flight.getTo());
}
```

About Spring and Hibernate/JPA

- Hibernate is one of the most powerful ORM technology used widely today
- Hibernate is 100% JPA compliant ORM
- Spring supports native Hibernate integration as well as integration via JPA
- The real benefit of Spring used along with Hibernate/JPA would be:
  - Delegating SessionFactory/EntityManagerFactory creation to Spring
  - Use DI wherever we need the SessionFactory/ EntityManagerFactory/EntityManager
  - **Manage transactions using Spring's declarative TransactionManager**

LTI
Let's Solve

Celebrating 20 Years

Spring MVC

Larsen & Toubro
Group Company

What is MVC?

- Well-established architectural pattern for dealing with UI
- **Model** manages the behavior and data of the application
- **View** renders the model into UI elements
- **Controller** processes user inputs and generates a response by operating on model objects

MVC in a Web Application

- The model is the data and business/domain logic for your application
- The view is typically HTML generated by your application
- The controller receives HTTP requests and decides which domain objects to use to carry out specific tasks

Benefits of MVC

- Decoupling views and models
- Reduces the complexity of your design
- Makes code more flexible
- Makes code more maintainable

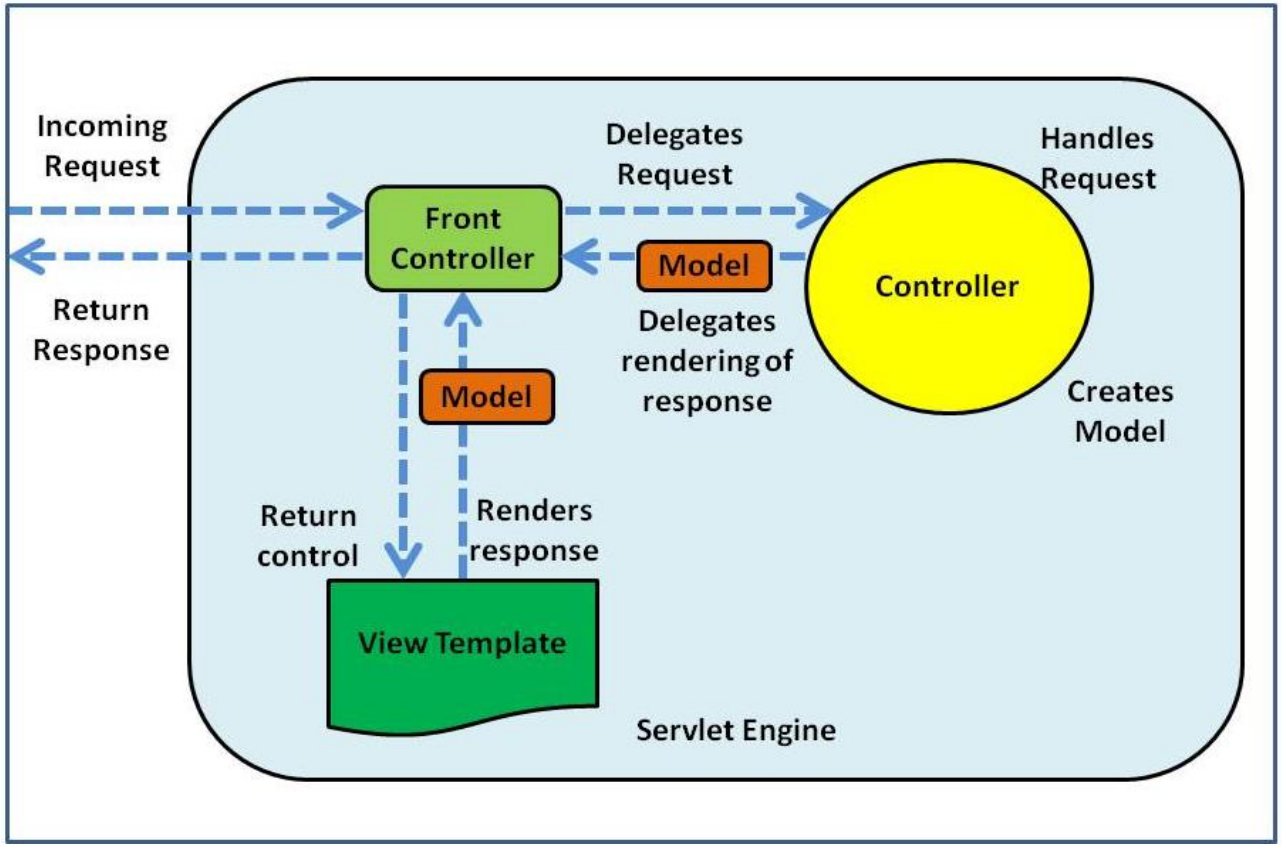
What is Spring MVC?

- MVC Web Framework
- Developed by the Spring team in response to what they felt were deficiencies in frameworks like Struts
- Deeply integrated with Spring
- Allows most parts to be customized (ie, you can use pretty much any view technology)
- RESTful functionality (URI templates, Content Negotiation)

Spring MVC Features

- Clear separation of roles
- Simple, powerful annotation-based configuration
- Controllers are configured via Spring, which makes them easy to use with other Spring objects and makes them easy to test
- Customizable data binding
- Flexible view technology
- Customizable handler mapping and view resolution

DispatcherServlet



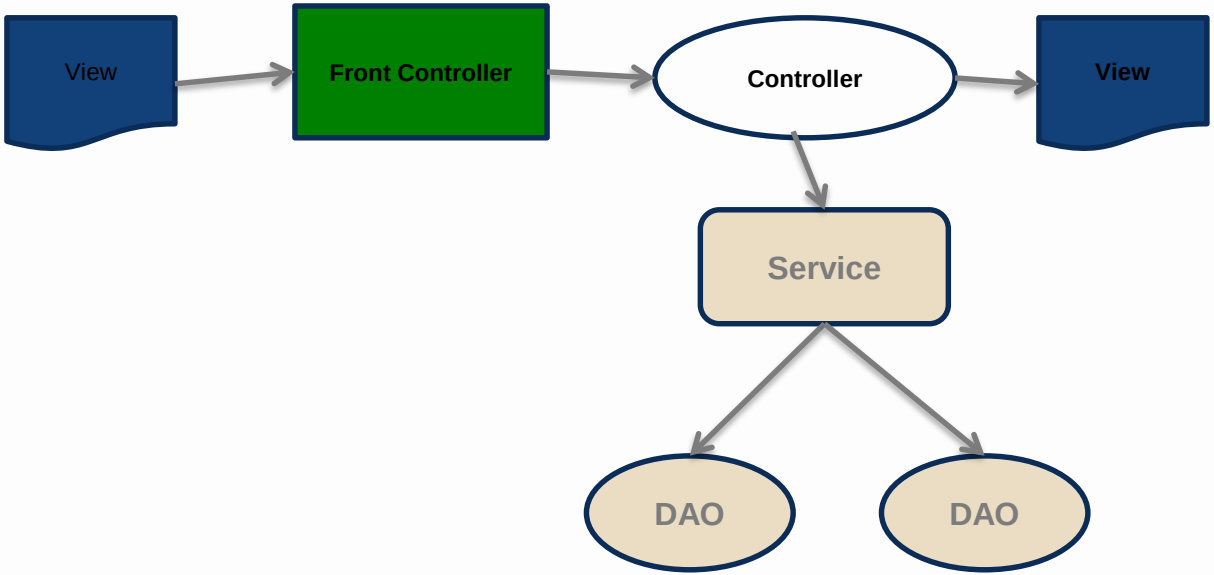
Controllers

- Spring MVC delegates the responsibility for handling HTTP requests to **Controllers**. Controllers are much like servlets, mapped to one or more URIs and built to work with HttpServletRequest and HttpServletResponse objects.

- The Controller API makes no attempt to hide its dependence to the Servlet API, instead fully embracing it and exposing its power.

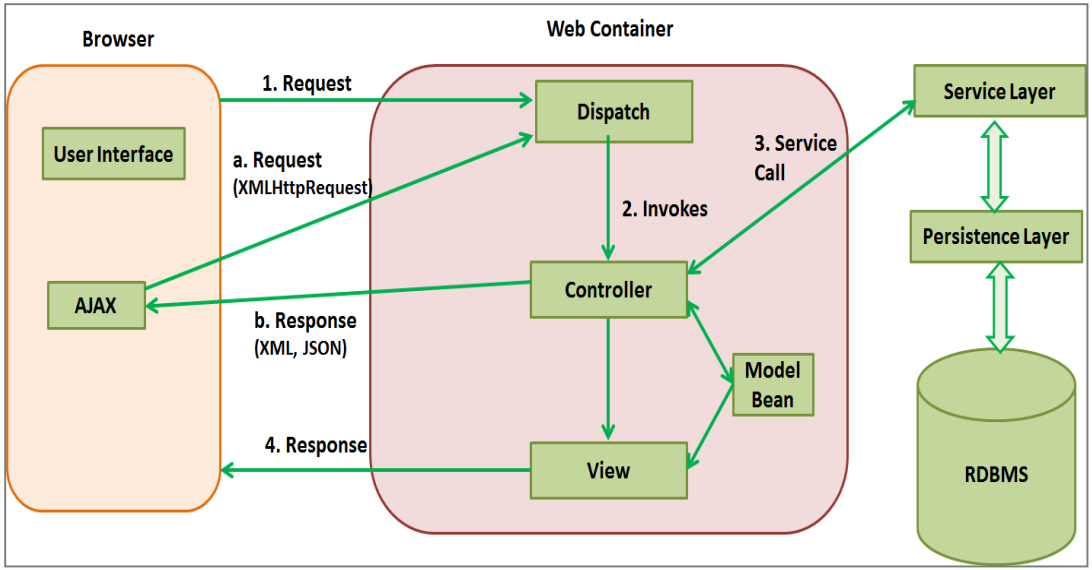
Cont'd…

- Controllers are responsible for processing HTTP requests, performing whatever work necessary, composing the response objects, and passing control back to the main request handling work flow. The Controller does not handle view rendering, focusing instead on handling the request and response objects and delegating to the service layer.



2 in 1

- Spring MVC can be used for developing traditional web applications as well as modern AJAX and RESTful applications as well



Simple Controller

- For most cases, you'll need to create a controller
- Create a class and annotate it with @Controller
- Then, create a method annotated with a @RequestMapping

```
package com.training;

@Controller
public class HelloController {

        @RequestMapping(value="/")
        public String hello() {
           return "hello";
        }
}
```

Advanced Request Mapping

- RequestMappings are really flexible
- You can define a @RequestMapping on a class and all method @RequestMappings will be relative to it.
- There are a number of ways to define them:
  - URI Patterns
  - HTTP Methods (GET, POST, etc)
  - Request Parameters
  - Header values


LTI
Let's Solve